AD _____

NAVORD REPORT 2741

SINGLE VS. TRIPLE ADDRESS COMPUTING MACHINES

22 JANUARY 1953

U. S. NAVAL ORDNANCE LABORATORY

WHITE OAK, MARYLAND

NAVORD Report 2741
Aeroballistic Research Report No. 149

SINGLE VS. TRIPLE ADDRESS COMPUTING MACHINES

Prepared by:

Calvin C. Elgot

ABSTRACT: The question: "Which is more desirable a single address or a triple address computing machine?" has been discussed, often with fervor from an engineering, economic, statistical and "personal preference" point of view. We limit our consideration to the question: "Which, of the two types of machines, requires fewer words to specify a sequence of instructions?" Utilizing a slightly idealized, (but we believe physically realizable), notion of a single address machine we obtain a partial answer to this question by proving a mathematical theorem, (page 5). Our result is embodied in Corollary 2, page 6. The strength of the result suggests that "in general," i.e. from a statistical point of view, fewer words are required to code by means of (idealized) single address than by triple address.

U. S. NAVAL ORDNANCE LABORATORY
WHITE OAK, MARYLAND

NAVORD Report 2741 22 January 1953

In this report there is advanced a basic formulation and solution of the problem "which electronic computer design is best, that with a multiple address system or that with a single address system." The study has been carried out as a part of task NR-044-003, Numerical Analysis and Theoretical Mechanics.

EDWARD L. WOODYARD
Captain, USN
Commander

H. H. KURZWEG, Chief
Aeroballistic Research Department
By direction

SINGLE VS. TRIPLE ADDRESS COMPUTING MACHINES

1. INTRODUCTION

By a machine word of a triple address machine we mean a sequence of four elements consisting of an operation code, first operand address, second operand address and storage address, respectively. By a machine half-word of a single address machine we mean a sequence of two elements, the first being an operation code, the second an operand address.

By an operation is meant an operator together with the numbers on which it operates ("+" is an operator; "1+2" is an operation). Assume a sequence of distinct arithmetic operations specified. Then the number of machine words, in a triple address machine, required to specify this sequence of operations is equal to the number of operations involved in the sequence. For example, if the sequence of operations is:

(1) $(a+b)$
(2) $(c.d)$
(3) $((c.d)-e)$
(4) $(((c.d)-e) / (a+b))$
(5) $(f|g)$; $(f|g) = g-f$
(6) $((f|g)\backslash(((c.d)-e) / (a+b)))$; $(x\backslash y) = (y/x)$

then six triple address machine words are required to specify this sequence.

We require our single address machine to have the property that the result of any particular operation is immediately available, (i.e. without any preliminary (programmed) shifting from one register to another), for possible use with the next operator. We assume that our single address machine is capable of performing inverse subtraction, (5), and inverse division, (6). Whether or not the triple address machine can perform these operations or not is inconsequential. We allow, too, the possibility of our machines performing unary operations.

We sometimes interpret (aob) as the result of applying the operator ob to the operand, (or argument), a. With this interpretation in mind we rather imagine our single address machine has a register, called, say, the argument register which houses the argument of the function which the machine is about to compute. When a function is computed the result is directed into the argument register where it is available for possible use with the next function. When a number is directed into the argument register it displaces any number which may have been there.

We say that an operation utilizes the preceding result if one of its operands is the preceding result. This situation is indicated in the table below by "+" under "P"; the contrary situation by "-" under "P."

The number of machine words (by definition equal to one half the number of machine half-words), required to specify one of a sequence of operations in a single address machine is given by the following table:

	P	S	W
(1)	+	-	$\frac{1}{2}$
(2)	+	+	1
(3)	-	-	1
(4)	-	+	$1\frac{1}{2}$

A "+" under "S" indicates the result of the operation is to be stored; a "-" indicates the contrary. Under "W" is given the corresponding number of machine words required to specify the operation in a single address machine. We see from the table a single address machine requires more or less machine words than a triple address machine to specify a sequence of operations according as the number of times the result of an operation is stored in more or less than the number of times the preceding result is utilized. We prove the first alternative does not hold, assuming the result of the last operation in the sequence is not stored.

We indicate the machine half-words which may be used to specify the sequence of six operations given above with our single address machine:

	Case of Table
(1) Transfer a to argument register. Add b to a. Store result.	(4)
(2) Transfer c to argument register. Multiply c by d.	(3)
(3) Subtract e from previous result.	(1)
(4) Divide previous result by (a+b). Store result.	(2)
(5) Transfer f to argument register Inverse subtract g from f.	(3)
(6) Inverse divide previous result by (4) result.	(1)

Each line above indicates a machine half-word. Thus five and one half words are required to specify this sequence of operations by means of a single address machine.

If T is the number of machine words required for a triple address machine,
if U is the number of machine words required for a single address machine,
if P is the number of times the preceding result is utilized,
if S is the number of times the result of an operation is stored then

$$T - U = \frac{1}{2}(P-S)$$

and this is true in general, (for arbitrary sequences of distinct operations). For each time a preceding result is utilized and the result not stored single address "gains" $\frac{1}{2}$ word over triple address, while if preceding result is not used and result is stored, single address "loses" $\frac{1}{2}$ word to triple address. In the other two cases there is no "gain" or "loss." We shall prove $P \geqslant S$.

2. TRANSITION TO FORMALITY

We introduce intermediate notions of SAO and TSAO for the purpose of making gradual the transition from ordinary" notation to the prefixed operator notation* which we employ. We define SAO as follows:

(1) A letter is an SAO.
(2) If F is a SAO and u a unary operator then u(F) is a SAO.
(3) If F and G are SAO's and o a binary operator then (FoG) is a SAO.
(4) A sequence of numbers and unary and binary operators is a SAO only when this follows from (1), (2), (3).

We define a transformed SAO, (TSAO), by means of rules (1), (2), (4), (with SAO replaced by TSAO and "(3)" replaced by "(3')"), and

(3') If F and G are TSAO's and b a binary operator then b(F,G) is a TSAO.

It is clear there is a biunique correspondence between SAO's and TSAO's.

For example $(\sqrt{(a+(b.c))}+d)$ is a SAO and the corresponding TSAO is $+(\sqrt{\ }(+(a,.(b,c))),d)$. The TSAO with parentheses and commas deleted is $+\sqrt{\ }+a\ .b\ c\ d$.

We claim that deleting all parentheses and commas in a TSAO creates no ambiguity in interpreting the "deleted TSAO." Otherwise stated if two distinct TSAO's are "deleted," the resultant sequences of symbols are distinct. A "deleted TSAO" is a word as defined below if we identify the operators with connectives (of the alphabet mentioned below) and use letters a_i, as below. The corollar to the theorem of the next section then justifies our claim. The "deleted TSAO" corresponding to the first example is:

$$\backslash\ |\ f\ g\ /\ -\ .\ c\ d\ e\ +a\ b$$

Order may be introduced into the "deleted TSAO," (thus inducing order into the TSAO and the SAO), and parentheses may be properly inserted to recover the original TSAO as follows:

Read from right to left until the first operator is encountered, then count two letters, (one letter if the operator is unary), from left to right and put parentheses around the two, (or one) letters separating the letters with a comma. The operator together with the letters is then reckoned as a single letter and the process is iterated. Using this order of operations we note that the result of an operation must be stored if and only if there is a next operation and the number of letters between the operator and the next, (from right to left),

*This notation is generally attributed to Lukasiewicz.

operator is two or one depending on whether the next operator is binary or unary. (This motivates the definition of long segment below.) An operator utilizes the previous result if and only if there is a previous operator and there are less than two or one letters between the operator and the previous operator depending on whether the operator is binary or unary. (This motivates the definition of short segment below.)

3. A SIMPLE LANGUAGE*

Let the alphabet consist of:

the letters - $a_1, a_2, a_3, \ldots$

the connectives - f_j^i ; $i, j = 1, 2, 3, \ldots$

A finite, possibly null, sequence of members of the alphabet is called a string. The length of the string is the number of elements of the sequence. If A is a string of length m and B a string of length n then AB is the string of length m + n whose i^{th} member, $1 \leq i \leq m$, is the i^{th} member of A and whose $(m+j)^{th}$ member, $1 \leq j \leq n$, is the j^{th} member of B. We define a word by induction on the length of a string. A string, S, is a word if and only if one of the following holds:

(1) $S = a_i$

(2) $W_1, W_2, .., W_n$ are words and $S = f_n^i W_1 W_2 \ldots W_n$ where i, n are positive integers.

The alphabet together with the rules of word formation and a suitable interpretation of words is a simple language.

The rank R(S) of a string S is defined as follows:

(1) $R(a_i) = -1$

(2) $R(f_n^i) = n - 1$

(3) If $S = S_1 S_2$, (S_1 , S_2 strings), $R(S) = R(S_1) + R(S_2)$

(4) The rank of the null string is zero.

If $S = S_1 S_2$ then S_1 is called a head of S. If S_2 is not null the head is called proper.

Theorem** A string S is a word if and only if the rank of every proper head of S is non-negative and the rank of S is -1.

---

*Rosenbloom, "The Elements of Mathematical Logic," pp 152-7.
**Ibid, p 154.

Corollary If $W = S_1 S_2$ is a word and S_2 is not null then there is one and only one word which is a head of S_2, (where S_1 and S_2 are strings).

4. THE THEOREM

In what follows our alphabet consists of the single letter a and the two connectives 1, 2 of degrees and one and two respectively.

Examples of Words

(1) 1 a

(2) 2 a a

(3) 2 1 a 2 a a

(4) 2 2 2 2 2 2 2 a a a a a a a 2 a a

Examples of Strings Which Are Not Words

(5) 1

(6) a 1 a

(7) 2 a 2 a a 2 a

We observe if W is a word then W = S a; if, further W ≠ a then W = nT where S and T are strings and n is a connective. (This is immediate from the definition.)

We define a segment of a string S as a string of the form mTn if S = U m T n V and m and n are connectives where U, V, T, are strings, T being a string, possibly null, of a's only. A segment S is called a long segment if and only if one of the following holds:

S = 1 a U

S = 2 a a U

where U is a string; a segment is called a short segment if it is not a long segment.

No segments occur in examples (1), (2), (5), (6). In example (3) S = 2 1 is a short segment, T = 1 a 2 is a long segment. In example (7) S = 2 a 2 is a short segment while 2 a a 2 is a long segment. In (4) S = 2 a a a a a a a 2 is a long segment. The "number of short (long) segments" shall always mean the number of short (long) segments counting multiplicity. In (4) there are six short segments.

Theorem In every word W the number of short segments is not exceeded by the number of long segments.

Proof We use induction on the length of W.

We denote the number of short segments occurring in W by W_s and the number of long segments by W_l. If W = a or W = 1 a or W = 2 a a the validity of the theorem is obvious. We have already observed and now we emphasize that if $W \neq a$ then W is of the form Sa = n T where n is a connective and S, T strings. We consider the following cases:

(1) $W = 1\ W'\ W \neq a$

(2) $W = 2\ W'W'',\ W' \neq a$

(3) $W = 2\ a\ W',\ W' \neq a$ where W,W',W'' are words.

Case (1): Clearly

$$W_s = 1 + W'_s \text{ while } W_l = W'_l$$

so that using the inductive assumption the theorem is valid in this case.

Case (2): $W_s \geq 1 + W'_s + W''_s$

$$W_l \leq 1 + W'_l + W''_l$$

Indeed equality actually holds in the first case and if $W \neq a$ also in the second case, as may be seen from the corollary above. The theorem follows again with the use of the inductive assumption.

Case (3): $W_s = 1 + W'_s$

$$W_l = W'_l$$

as before the theorem follows which completes the proof.

Corollary 1 If $W \neq a$ then

$$T \geq 2S + 1$$

$$T \leq 2P + 1$$ where S = number of long segments of W

P = number of short segments of W

T = total number of connectives in W

Proof If $W \neq a$ then

$$P + S + 1 = T$$

Corollary 2 In the notation of page 2, $T \geq U$.

Proof $P = W_s$, $S = W_1$ from paragraph immediately preceding section 3; $T - U = \frac{1}{2}(P-S)$ from page 2 and $W_s \geq W_1$ from the theorem.

Remarks

(1) The theorem immediately above fails if we allow connectives of degree greater than two. Indeed if $n > 2$ is a connective of degree n then

n 2 a a 2 a a . . . 2 a a, (2 a a occurring n times),

is a word with one short segment and (n-1) long segments, (extending the notion of long and short segment in obvious fashion).

(2) If a sequence of, (not necessarily distinct), operations are given we associate with it a sequence of SAO's as follows:

Of all "sub-SAO's" which occur more than once we seek one of minimum "length" and record it, replacing all occurrences of this SAO by a single letter. If there are other "multiple occurring" SAO's of the same "length" we record them one by one replacing all occurrences of each by distinct letters distinct from the first. This process is iterated until the remaining SAO consists of distinct operations. We then have a sequence of distinct sequences of distinct operations (in a slightly extended sense since we are now admitting letters). If R is the "length" of the sequence of sequences then our result modifies to

$$\frac{1}{2}(R-1) + P \geq S$$

for the result of each SAO in the sequence of SAO's must be stored, except for the last.

(3) A dichotomous conditional transfer does not require more single address machine-words than triple address machine-words while a trichotomous conditional transfer may.

(4) As a by-product of the techniques employed one notes that a machine could be built, (or a subroutine on an existing machine could be constructed), which would be capable of interpreting a sequence of symbols as a sequence of arithmetic operations and capable, too, of deciding by itself when and where the result of an operation must be stored.

(5) The ordered SAO obtained by introducing order into the SAO in accordance with the rule given on page 2, may be regarded formally as a sequence of SAO's. Every ordered SAO corresponds to a sequence of arithmetic operations but the converse is not true. This fact doesn't effect our conclusion since the machine word requirement in a triple address machine does not depend on the order in which the operations are performed. It is easy to see, however, that a necessary and sufficient condition for a sequence of arithmetic operations, written say as on page 1, to correspond to an ordered SAO is that every binary operator utilizing one (or two) previous result(s) as operand(s) must obtain the result(s) from the previous first (and second) line(s) and every unary operator utilizing a previous result must obtain it from the preceding line.

(6) In stating the conditions, (top page 4), under which the preceding result is utilized we have allowed replacing operations like xoy by yOx if the two are equal. (Here "o" and "O" stand for operators.)

If we replace each xoy, with y the previous result, by yOx in the given ordered SAO and then write the corresponding TSAO, the corresonding "deleted TSAO" and the corresponding word, then all short segments of this word will be of the form 22, 21, 12, or 11. (None will be of the form 2a2 or 2a1.)